A State Event Detection Algorithm for Numerically Simulating Hybrid Systems with Model Singularities

JOEL M. ESPOSITO

US Naval Academy, Annapolis, Maryland

VIJAY KUMAR

University of Pennsylvania, Philadelphia, Pennsylvania

esposito@usna.edu, kumar@grasp.cis.upenn.edu

This paper describes an algorithm for detecting the occurrence of events, which signify discontinuities in the first derivative of the state variables, while simulating a set of non-smooth differential equations. Such combined-discrete continuous systems arise in many contexts and are often referred to as hybrid systems, switched systems, or non-smooth systems. In all cases the state events are triggered at simulated times which generate states corresponding to the zeros of some algebraic "event" function. It has been noted that all existing simulators are prone to failure when these events occur in the neighborhood of model singularities – regions of the state space where the right-hand side of the differential equation is undefined. Such model singularities are often the impetus for using non-smooth models in the first place. This failure occurs because existing algorithms blindly attempt to interpolate across singular regions, checking for possible events after the fact. The event detection algorithm described here overcomes this limitation using an approach inspired by feedback control theory. A carefully constructed extrapolation polynomial is used to select the integration step size by checking for potential future events, avoiding the need to evaluate the differential equation in potentially singular regions. It is shown that this alternate approach gives added functionality with little impact on the simulation efficiency.

Categories and Subject Descriptors: I.6.8 [**Simulation and Modeling**]: Types of Simulation–*combined discrete-continuous*

General Terms: Algorithms

Additional Key Words and Phrases: Hybrid systems, model singularities, event detection, numerical integration, discontinuities

1. INTRODUCTION

Many engineering systems are described by sets of differential equations with discontinuous right-hand sides. A very basic example is as follow. For $x : t \in \mathbb{R}^+ \rightarrow$

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JAN 2007		**00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
A State Event Detection Algorithm for Numerically Simulating Hybrid Systems with Model Singularities	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **United States Naval Academy,Annapolis,MD,21402**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
ACM Transactions on Modeling and Computer Simulation, Volume 17, Issue 1, p. 1-22, January 2007

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**26**	

$x(t) \in \mathbb{R}^N$, we have

$$\dot{x} = \frac{dx(t)}{dt} = \begin{cases} f^1(x), & \text{if } g^1(x,t) < 0 \\ f^2(x), & \text{if } g^2(x,t) \equiv -g^1(x,t) \leq 0, \end{cases} \tag{1}$$

where $g^1(x,t) = -g^2(x,t)$ ensures the conditions are mutually exclusive, although significantly more complex switching structures are permitted. When the algebraic *event function* $g^i(x(t),t)$ (also called a guard or discontinuity function) changes signs an *event* occurs, causing a discontinuity in the first derivative of the state as the derivative definition $f^i(x)$ switches values of the discrete variable, $i = 1$ to $i = 2$. The abrupt switches in i resemble discrete-event systems, while the dynamics of x are governed by continuous differential equations. Such systems arise in many contexts such as contact mechanics, phase transition and failure mode simulation, and are often referred to as combined discrete/continuous dynamic systems, hybrid systems, switched systems, or simply non-smooth systems. Of particular interest is the fact that physical systems controlled by embedded microprocessors can be modelled in such a fashion (see Tomlin and Greenstreet [2002] and others in that series).

It has been shown [Cellier 1979] that the proper method of simulating such systems is through the *discontinuity locking approach*, which requires the definition of the derivative remain "locked" during each integration step. After a step is taken, it must be determined if it is possible that $g^i(x,t)$ changed sign on the integration interval. If so, the precise time, t^*, at which the event occurred must be determined (i.e., $g^i(x(t^*),t^*) = 0$) so that the integration process can be restarted using the new function for the derivative, using $x(t^*)$ as the initial condition. Failure to properly detect and locate events can have disastrous ramifications on simulation fidelity [Branicky 1995], and the ability to detect events accurately is a key attribute of quality simulators [Mosterman 1999]. Unfortunately the event detection problem is symbolically undecidable [Ruohonen 1994]. The problem is also hard computationally [Shampine et al. 1991].

The algorithm in this paper addresses a subtle but potentially disastrous shortcoming of even the best existing algorithms. Even the most sophisticated class of methods (which we collectively term *interpolation-based* methods) fail to localize an event which occurs in the neighborhood of a model singularity – a region of the state space where the derivative function $f^i(x,t)$ is undefined. To quote from

Park and Barton [1996]: "(the discontinuity locking) ... approach has been demonstrated to be both efficient and correct *provided* the system of equations employed before the state event is mathematically well behaved in a small interval following the event, even if the solution is not physically meaningful." In this paper we introduce an event detection scheme which overcomes these pitfalls by using carefully crafted *extrapolation* polynomials along with control-theoretic techniques to select the integration step size based on proximity to the event surface. The algorithm possesses the special property that it only approaches the switching surface from one side. This ensures it never accidentally evaluates the model at a singularity. It is shown here that this alternate approach gives added functionality with little impact on the simulation efficiency.

Differential equations in which the domain of the right hand side of the ODE is *not* all of $\mathbb{R}^N$ appear in a variety of contexts in which the state space of a dynamical system is a manifold with boundary. In certain situations, the vector field is not defined everywhere because the model was only designed to be applicable in certain regions of the state space. For example, in an aerial vehicle model it may not be possible to evaluate the supersonic aerodynamic model at sub-sonic velocities. In other situations the singular region is not a modelling artifact but rather dictated by the physics of the problem. For example in chemical reaction problems, where the state variables are the concentrations of various chemical species, negative values of the state are physically meaningless but may be introduced due to numerical errors in the simulation. Similar situations arise in problems where phase transitions occur. Finally, in feedback control systems, the control law itself may be constructed in such a way that it cancels or inverts the nonlinear dynamics of the system. Such controllers are known to have singularities in regions of the state space where the dynamics are not invertible, in which case the control computer should switch to a different feedback law in this region. An example of such a controller is one which depends on the results of an inverse kinematics solver for a robot arm. If the requested end-effector configuration is infeasible, a solution for the inverse kinematics does not exist. Other controllers which rely on inverting the Jacobian are inapplicable at singular configurations inside the workspace. In general any differential equation whose right hand side involves a square root, logarithm, inverse trigonometric function, or division by a state variable will posses a singularity.

Such events are often referred to as *unilateral events*, meaning that they should be detected without actually allowing the simulation to progress to the point where the state variable crosses the event surface.

In Section 1.1 we establish notation and provide some background on the problem. In Section 1.2 we examine other event detection algorithms and their shortcomings in more detail. In Section 2, we present an overview our algorithm. Section 3 presents three example problems which demonstrate the method's effectiveness, robustness, and efficiency, respectively. Concluding remarks are presented in Section 4.

1.1 Notation and Background

Formally, assume the state $x \in \mathcal{X} \subset \mathbb{R}^N$ and that $t \in \mathbb{R}^+$. The i^{th} definition of the derivative $f^i(x)$ is defined for $x \in \mathcal{X}^i \subset \mathcal{X}$ as $f^i : \mathcal{X}^i \rightarrow \mathbb{R}^N$, and is continuous in $x \in \mathcal{X}^i$. It is assumed that the function $g^i : \mathcal{X}^i \times \mathbb{R}^+ \rightarrow \mathbb{R}$ is smooth in x and t. Initially i is set such that $g^i(x(t_0), t_0) < 0$. The state evolves according to $\dot{x} = f^i(x)$ until $g^i(x,t) \geq 0$, which means the value of i changes to i' according to the switching structure the model possesses. Once i changes to i', a new definition for $\dot{x}$ must be used, $f^{i'}(x)$, and a new event function $g^{i'}(x,t)$ is employed. We refer to each value of i, and the associated f^i and g^i as a system *mode*. Although various formalisms, such as hybrid automata, for modelling such systems exist, we prefer to use the general notion introduced here to emphasize that our simulation technique is applicable to any system with this structure. Throughout this paper we refer to any state/time pair x, t such that $g^i(x,t) = 0$ as an *event*. We refer to $g^i(x,t)$ as the *event function*. For a given value of t, the level set defined as the set of all $x \in \mathcal{X}^i$ such that $g^i(x,t) = 0$ is called the *event surface*.

For the remainder of this paper we will discuss numerical integration schemes based on difference equations that produce estimates of the state at a discrete set of times t_k, $k = 0, 1, 2, \ldots$. We say that the subscript k indexes the iteration number. Let the integration step size be defined as $h_{k+1} = t_{k+1} - t_k$. For clarity $x_k = x(t_k)$ and $f_k = f(x_k)$ and $g_k = g(x_k, t_k)$.

Almost all of the algorithms reviewed in the following section are loosely based on the discontinuity locking approach [Cellier 1979] combined with the use of interpolation polynomials (we refer to them as *interpolation-based* methods) so it is important to understand the basic high-level outline of these algorithms (see Fig-

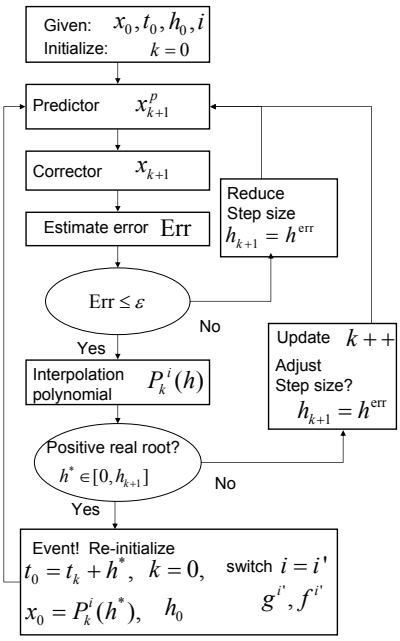

Fig. 1. The traditional simulation algorithm. The event dynamics are not used to select the step size for the next iteration.

ure 1), to understand why they fail in the neighborhood of model singularities. During the k^{th} simulation iteration the algorithm:

1. integrates the current state estimate x_k, using $f^i(x)$, through a time step h_{k+1}, producing a new state estimate x_{k+1};
2. constructs a polynomial $P_k^i(h)$ which *interpolates* the event function $g^i(x(t_k + h), t_k + h)$ over $h \in [0, h_{k+1}]$, whose accuracy is on the same order as the underlying integration algorithm;
3. determines if $P_k^i(h)$ has roots in the interval, if so computes the smallest positive root h^*, implying $g^i(x(t_k + h^*), t_k + h^*) = 0$;
4. if no real positive roots exist, $k = k + 1$ and the integration continues in Step 1. Else, an event has occurred, the algorithm updates $i \rightarrow i'$, switches the right hand side of the ODE to $f^{i'}(x)$, the event function to $g^{i'}(x, t)$, and restarts integration using $x(t^*)$ where $t^* = t_k + h^*$ as the new initial conditions.

Note that the integration is performed first, without any consideration to the value of the event function or proximity to the boundary of $\mathcal{X}^i$, typically only truncation error estimates are used to select h_{k+1}. Therefore if at the beginning of an iteration the state is close to the boundary of the domain of $f^i(x,t)$ it is entirely possible for Step 1 to generate a state, x_{k+1}, which lies outside the domain of $f^i(x)$ – a potentially singular region. Because most integration methods evaluate the derivative of the state at the integration interval endpoint, the simulation will crash before ever reaching Steps 2 and 3, where event considerations come into play. Therefore regardless of the level of sophistication used to perform Steps 2 and 3, the algorithm is doomed to failure in such a situation.

1.2 Related Work

The first to note that events whose event function depend on the state of the system warrant special treatment was Cellier [1979], who also introduced the discontinuity locking approach still used today in nearly all the work mentioned in this section. The first work to notice that the rate of change of the event function along the flow field, was a critical quantity in event detection was Carver [1978]. This derivative is defined as

$$\frac{dg^i}{dt} \equiv \frac{\partial g^i(x,t)}{\partial x} \cdot f^i(x) + \left.\frac{\partial g^i(x,t)}{\partial t}\right|_{(x,t)=(x_k,t_k)}, \tag{2}$$

where $(\cdot)$ denotes the dot product of two N-dimensional vectors defined in the usual sense. The idea of differentiating the event function and appending it as an extra state variable to be integrated, thus rendering a new event function which is linear in the extended state, was introduced there as well. However, in these early works as well as in Hay and Griffin [1979], Joglekar and Reklaitis [1984], and Prestin and Berzine [1991], events were detected by simply looking for sign changes in the event function (or perhaps its derivative) after integrating through one time step. As a result they may fail to detect an event when multiple sign changes of the event function occur in the course of a single simulation step.

Recent methods emphasize guarantees that all events in a given time step are properly detected using interpolation polynomials. The fact that interpolation polynomials can be generated for the event function dynamics as a by-product of the integration process was first exploited in Shampine et al. [1991]. Using these interpolants, they are able to correctly identify even multiple event occurrences using

Strum sequences so long as the guards are polynomial expressions. However, this information is not used to select step sizes. This technique is the first method which can be demonstrated to be "correct"; meaning it is capable of detecting multiple events in a given integration interval. More recently, in Park and Barton [1996] some of these ideas are combined and methods from interval arithmetic are used to create efficient "exclusion tests." Exclusion tests are designed to determine if it possible that a polynomial has a root in a given interval using minimal computation. They are guaranteed to not return false negatives so any interval which passes the test may be removed from further consideration, while intervals that do not pass they test are subject to further scrutiny (and computation time) to determine the location of the potential roots. This event detection method seems to be the most reliable technique in the literature to date, and it is streamlined and well suited to stiff problems or differential algebraic equations. This work represents a much more computationally efficient variant of the algorithm in Shampine et al. [1991], targeted at improving the robustness of differential algebraic equation simulation. Similar work, such as Bahl and Linninger [2001], focused on developing other exclusion tests. Again, each of these methods fails to locate events which are close to model singularities because they attempt to evaluate the right-hand side of the differential equation *before* determining if an event (or singularity) has occurred.

In related but orthogonal work Esposito and Kumar [2004] address reliable event detection for so-called multi-agent systems. There the emphasis is on improving computational efficiency of large-scale simulations through asynchronous integration schemes. We note that some of the step-size selection techniques used here are inspired by analogies to automatic feedback control systems, where proximity to the event surface is treated as an output variable. In a similar vein Gustafsson [1994] introduces a control theoretic step size selection scheme in which the truncation error is treated as an output variable.

2. A NEW EVENT DETECTION ALGORITHM

In contrast to the approach outlined above, which integrates first and looks for events using an interpolant afterward, the approach advocated here is as follows. At the k^{th} iteration:

1. Construct a polynomial $P_k^i(h_{k+1})$, whose accuracy is on the same order as the

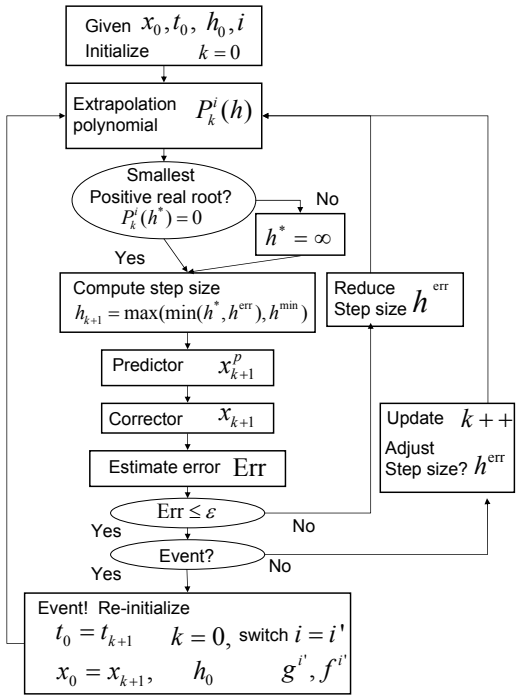

Fig. 2. The new simulation algorithm introduced in this paper. The extrapolation polynomial enables the computation of a step size based on proximity to the event surface.

underlying integration algorithm, that *extrapolates* the event function $g^i(x(t_k + h_{k+1}), t_k + h_{k+1})$ over $h_{k+1} \in [0, \infty)$.

2. Determine if the extrapolation polynomial $P_k^i(h_{k+1})$ has a positive real root h^* – corresponding to an event; if so, compute the smallest such root h^*. If no positive real roots exist, $h^* = \infty$. In addition, a traditional algorithm can be use to select a step size, h^{err}, based on truncation error estimates.
3. Integrate the state x_k through a step $h_{k+1} = \min(h^*, h^{\text{err}})$ producing x_{k+1}.
4. if an event occurred, $g^i(x_{k+1}, t_{k+1}) = 0$, update $i \rightarrow i'$, switch the right-hand side of the ODE to $f^{i'}(x)$ and event function to $g^{i'}(x, t)$ and restart integration using $x(t_{k+1})$, where $t_{k+1} = t_k + h_{k+1}$, as the new initial condition.

Graphically the algorithm is depicted in Figure 2. In the remainder of this section we show how to construct the polynomial in Step 1, and present the algorithm in

detail.

2.1 Preliminaries

Because the mode remains fixed during a given integration step we replace $g^i(x,t)$ and $f^i(x)$ with $g(x,t)$ and $f(x)$ for the purpose of streamlining notation when possible.

Since many models of interest are linear, we consider this to be the most important class of event function. In addition, note that it is possible to transform *any* nonlinear guard to a linear one by adding an extra state variable, $z = g(x,t)$

$$\left\{ \begin{array}{c} \dot{x} = f(x) \\ \\ g(x,t) \leq 0 \end{array} \right\} \leftrightarrow \left\{ \begin{array}{c} \dot{x} = f(x) \\ \dot{z} = \frac{\partial g}{\partial x} \cdot f(x) + \frac{\partial g}{\partial t} \\ z \leq 0. \end{array} \right\} \tag{3}$$

In the remainder of this paper we assume the event function is linear unless otherwise stated.

Regarding the selection of integration method, we cannot use a purely implicit integration method because it requires evaluating $f(x)$ at a possibly singular endpoint; on the other hand strictly explicit integration schemes are known to have poor stability and accuracy. Instead our method of choice is the predictor-corrector approach (see for example Benan et al. [1989]) since we can use the explicit predictor component to test for events and the semi-implicit corrector component to improve stability and accuracy. For the predictor-corrector numerical integration method, the predictor equation is

$$x^p_{k+1} = x_k + h_{k+1}\{\sum_{j=1}^{m} \beta^p_j(h_{k+1}) f_{k-j+1}\}, \tag{4}$$

the result of which is used in the corrector equation to generate x_{k+1}

$$x_{k+1} = x_k + h_{k+1}\{\beta^c_0(h_{k+1}) f(x^p_{k+1}) + \sum_{j=1}^{m-1} \beta^c_j(h_{k+1}) f_{k-j+1}\}. \tag{5}$$

Recall that the β's are polynomial weighting functions in h_{k+1} in the case of non-constant step sizes. Therefore the event dynamics after the predictor and corrector phases are, respectively,

$$g^p_{k+1} = g(\, x_k + h_{k+1}\{\sum_{j=1}^{m} \beta^p_j(h_{k+1}) f_{k-j+1}\},\ t_k + h_{k+1}\,), \tag{6}$$

$$g_{k+1} = g(\, x_k + h_{k+1}\{\beta_0^c(h_{k+1})f(x_{k+1}^p) + \sum_{j=1}^{m-1} \beta_j^c(h_{k+1})f_{k-j+1}\},\ t_k + h_{k+1}\,). \quad (7)$$

Because the final value of g_{k+1} depends on the corrector dynamics which, in turn, require evaluating $f(x)$ at x_{k+1}^p (a potential singularity), we must base the computation of h_{k+1} on the predictor dynamics alone and hence eq. (6).

2.2 Constructing the Extrapolation Polynomial

Using a Taylor series expansion of the predicted event function eq.(6)

$$g_{k+1}^p = g_k + h_{k+1}\frac{dg^p}{dt}\bigg|_{(x,t)=(x_k,t_k)} + \frac{h_{k+1}^2}{2!}\frac{d^2g^p}{dt^2}\bigg|_{(x,t)=(x_k,t_k)} + \ldots, \quad (8)$$

we can determine the value of g_{k+1}^p as a function of the, yet undetermined, step size h_{k+1}. Recalling that it was assumed that the event function is both linear in t and $x(t)$, we can neglect the second and higher order terms in the expansion. In addition, the first-order partials of $g(x,t)$ are constant everywhere so we drop the designation of where they are evaluated. So, by the chain rule,

$$\frac{dg^p}{dt} = \frac{\partial g^p}{\partial x}\cdot\dot{x} + \frac{\partial g^p}{\partial t} \quad (9)$$

and, within the integration error tolerance, the predictor equation gives

$$\dot{x} \approx \sum_{j=1}^{m} \beta_j^p(h_{k+1})f_{k-j+1}. \quad (10)$$

Therefore, utilizing the linearity assumption, and substituting eqs. (10) and (9) into eq.(8),

$$g_{k+1}^p(h_{k+1}) = g_k^p + h_{k+1}\left\{\frac{\partial g^p}{\partial x}\cdot\sum_{j=1}^{m}\beta_j^p(h_{k+1})f_{k-j+1} + \frac{\partial g^p}{\partial t}\right\} \quad (11)$$

which determines g_{k+1}^p as a function of the step size, up to the integration error tolerance.

THEOREM 2.1. *Selecting*

$$h_{k+1} = \frac{(\gamma-1)g_k^p}{\frac{\partial g^p}{\partial x}\cdot\sum_{j=1}^{m}\beta_j^p(h_{k+1})f_{k-j+1} + \frac{\partial g^p}{\partial t}}, \quad (12)$$

where $\gamma \in [0,1)$, causes the event dynamics to behave like a stable linear system which exponentially converges to the surface $g(x,t)=0$; furthermore, if $g(x_0,t_0) < 0$ then $g(x_k,t_k) \le 0, \forall k$.

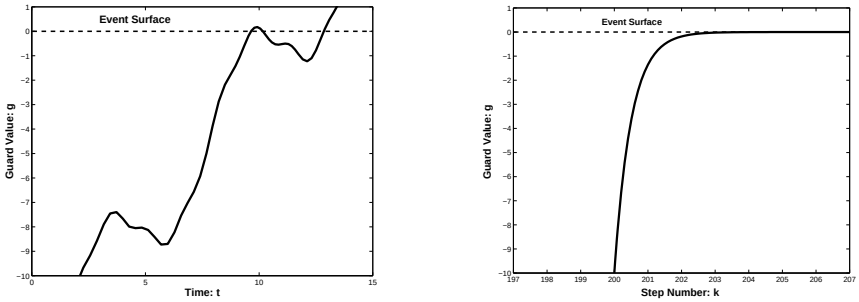

Fig. 3. The natural event dynamics as a function of time (*left*). After choosing h_{k+1} according to eq.(12), the event dynamics as a function of the iteration number, k, have been reparametrized to behave like a stable linear system (*right*).

PROOF. Substituting eq. (12) into eq. (11), yields

$$g^p_{k+1}(h_{k+1}) = \gamma g^p_k. \tag{13}$$

The solution to this difference equation is

$$g^p_k = \gamma^k g^p_0. \tag{14}$$

As long as $\|\gamma\| < 1$, $g_k \to 0$ as $k \to \infty$. In addition $\gamma \geq 0$, implies that g^p_k does not change sign. Therefore, assuming that $g_0 < 0$ then $g_k \leq 0$, $\forall k$, ensuring that the state never crosses over to the potentially singular region $g(x_k, t_k) > 0$, which is the desired effect. Figure 3 illustrates the effect of this choice of step size. □

Remark 2.2. Note that since β^p_j on the right-hand side of eq. (12) is a polynomial function of h_{k+1}, solving for h_{k+1} requires a root finding algorithm and eq. (12) is essentially an extrapolation polynomial, whose roots determine possible event locations

$$P^i_k(h_{k+1}) = h_{k+1}\left(\frac{\partial g}{\partial x}\cdot\sum_{j=1}^{m}\beta^p_j(h_{k+1})f_{k-j+1} + \frac{\partial g}{\partial t}\right) + (1-\gamma)g^p_k, \tag{15}$$

Remark 2.3. Extending this method to non-linear event functions is straightforward when the event function has a Taylor series expansion of finite length (*i.e.*, polynomials). Then h_{k+1} must be selected to cancel the higher order terms in the expansion as well.

Remark 2.4. The method is easily extensible to Boolean combination of such functions [Esposito 2002] (*i.e.*, polyhedral and semi-algebraic sets).

Remark 2.5. Transcendental event functions are more difficult to deal with. An exact cancellation of all higher order terms is not possible. Instead the dominant terms can be cancelled, but the uncancelled higher order terms make it impossible to guarantee $g(x_k, t_k) \leq 0 \ \forall k$. In practice, a conservative value of γ (*i.e.* values significantly greater than zero) can be used to mitigate the effect of the uncancelled terms and improve performance.

Remark 2.6. It is interesting to consider an analogy to feedback control systems. We consider the event function to be the *output* variable and the step size h_{k+1} to be the *input* variable. Eq. (11) represents the dynamics of the system. Eq. (12) represents a control law and γ is a "gain". In this context h represents the speed with which the state approaches the event surface. The technique employed is similar to a nonlinear control technique called feedback linearization [Isidori 1995]. Note that in control system parlance, the control law produces a response with *no overshoot*, meaning $g_k^p \leq 0, \forall k$.

2.3 Selecting γ

The rate at which the state converges to the event surface is determined by the value of the gain, $\gamma \in [0, 1)$. Values closer to zero produce rapid convergence while values closer to one cause g_k^p to approach zero more slowly. Note that if our only goal was to drive $g_k^p \to 0$, in as few iterations as possible γ should be set to zero. Instead we must drive the *corrected* value, g_{k+1} in eq. (7), to zero and, if we are to avoid any possible singularities, ensure that $g_{k+1} \leq 0 \ \forall k$. In this section we introduce a method of selecting γ which leverages the existing truncation error control mechanism.

While it is impossible to know the discrepancy between the predicted and corrected state *a priori*, it is possible to compute bounds. Recall that the truncation error control mechanism selects step sizes in such a way that the estimated integration error remains just below a user defined threshold. The error is estimated using Milne's method [Ascher and Petzold 1998]

$$\text{Err} \approx C \| x_{k+1}^p - x_{k+1} \| \tag{16}$$

where C is a known constant which is unique to the particular predictor-corrector method. If ϵ is the user defined maximum error threshold, then a particular step is accepted if $\text{Err} \leq \epsilon$ and the integration proceeds. If instead $\text{Err} > \epsilon$, the result is

rejected, a smaller step size is used, and the state is recomputed. Thus one knows that in any *acceptable* integration step

$$\|x^p_{k+1} - x_{k+1}\| \leq (\epsilon/C), \tag{17}$$

which implies in the worst case

$$g(x_{k+1}, t_{k+1}) = g(x^p_{k+1} + (\epsilon/C)\hat{u}, t_{k+1}), \tag{18}$$

where $\hat{u}$ is some unknown unit vector in $\mathbb{R}^N$. Obviously, for linear guards, the direction for $\hat{u}$ which produces the largest change in $\|g(x^p_{k+1}, t_{k+1}) - g(x_{k+1}, t_{k+1})\|$ is parallel to $\frac{\partial g}{\partial x}$. Therefore, even if the dynamics of the corrector cannot be precisely determined in advance, the following inequality holds

$$g(x^p_{k+1}, t_{k+1}) - (\epsilon/C)\|\frac{\partial g}{\partial x}\| \leq \ g(x_{k+1}, t_{k+1}) \ \leq g(x^p_{k+1}, t_{k+1}) + (\epsilon/C)\|\frac{\partial g}{\partial x}\|, \tag{19}$$

where the partial derivatives are evaluated at $(x, t) = (x_k, t_k)$. Thus, $(\epsilon/C)\|\frac{\partial g}{\partial x}\|$ is a natural bound on how accurately one can "control" $g(x_{k+1}, t_{k+1})$, and therefore represents the accuracy with which one can locate the event occurrence. Not surprisingly this bound depends critically on ϵ, the user determined integration accuracy. Some techniques claim to locate the event with greater precision; however it is meaningless to locate the event within a tighter tolerance than the accuracy with which state estimates are generated (the integration tolerance) [Shampine et al. 1991].

Given this bound, γ can be selected in such a way as to produce the fastest possible convergence consistent with those bounds and the nature of the system being simulated. We divide event detection tasks into three categories: unilateral events, bilateral events, and accuracy critical events. This distinction has been made elsewhere in the literature [Bahl and Linninger 2001]. The distinction is a necessary one because it is impossible to locate the time at which $g(x(t), t) = 0$ precisely, due to the fact that simulations are computed with finite-precision arithmetic.

Situations in which a model singularity exists, or in which the physics of the problem dictate that the state should *never* cross the event surface fall into the category of *unilateral* events. Indeed it is this class of events the method presented here is targeted at; and it is also this class of events which solicits failure in traditional event detection techniques. It is possible to guarantee that $g_{k+1} < 0, \ \forall k$

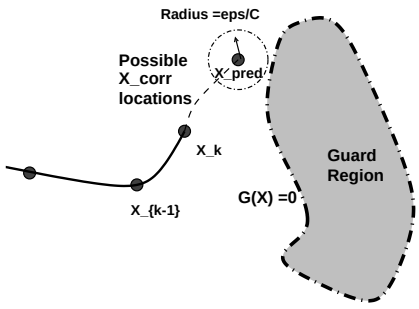

Fig. 4. A graphical illustration of the selection of γ in eq.(20) for unilateral events. The corrected state is guaranteed to lie within a circle of radius ϵ/C and therefore *not* cross the event surface.

(*i.e.* one-sided convergence) by selecting γ

$$0 \leq -(\epsilon/C)\frac{\left\|\frac{\partial g}{\partial x}\right\|}{g_k} \leq \gamma < 1. \tag{20}$$

This is illustrated geometrically in Figure 4. Using the linearizing step size in eq. (12) along with the minimum value of γ prescribed by eq. (20), results in the following guard dynamics

$$g_{k+1}^p = -\left(\frac{\epsilon}{C}\right)\left\|\frac{\partial g}{\partial x}\right\|\Big|_{(x,t)=(x_k,t_k)}. \tag{21}$$

According to eq. (19) this is the closest point to the event surface at which x_{k+1}^p can lie to be assured that $g(x_{k+1}) \leq 0$, if the integration is performed at tolerance ϵ.

Remark 2.7. In applications in which no model singularities exist and the state is permitted to possibly cross the event surface, accuracy and efficiency are the chief concerns. Then $\gamma = 0$ produces the most rapid convergence. In other applications where the state *must* cross the event surface in order to properly trigger the event and prevent "discontinuity sticking," an alternative selection is $\gamma = (\epsilon/C)\frac{\|\frac{\partial g}{\partial x}\|_{(x,t)=(x_k,t_k)}}{g_k}$. This approach is similar in spirit to Park and Barton's "consistent event location" method, which insures events are properly triggered by shifting the guard surface to the "right" of $g = 0$ by an amount equal to the integrator's truncation error tolerance.

2.4 Step Size Computation and Simulation Algorithm

Based on the developments in the previous sub-sections we present the proposed simulation algorithm in detail here (see also Figure 2). In addition we present prac-

Given x_0, t_0, h_0, a minimum acceptable step size $h^{\min}$, an initial value for h^{err}, a desired truncation error ϵ, an initial mode i and a termination time t^{final}. Set $k = 0$.
repeat
  Mode Set $g(x,t) \leftarrow g^i(x,t)$, $f(x) \leftarrow f^i(x)$
  Construct Extrapolation Polynomial

$$P(h_{k+1}) = h_{k+1}\left(\frac{\partial g}{\partial x}\cdot\sum_{j=1}^{m}\beta_j^p(h_{k+1})f_{k-j+1}+\frac{\partial g}{\partial t}\right)+(1-\gamma)g_k^p$$

  where $\gamma = -(\epsilon/C)\frac{\|\frac{\partial g}{\partial x}\|\,|_{(x,t)=(x_k,t_k)}}{g_k}$.
  Roots r = real positive roots of $P(h_{k+1})$
  if $r = \emptyset$ **then**
    $h^* = \infty$.
  else
    $h^* = \min(r)$.
  end if
  Select Step Size $h_{k+1} = \max[\min(h^*, h^{\text{err}}), h^{\min}]$
  Predictor $x_{k+1}^p = x_k + h_{k+1}\{\sum_{j=1}^{m}\beta_j^p(h_{k+1})f_{k-j+1}\}$
  Corrector $x_{k+1} = x_k + h_{k+1}\{\beta_0^c(h_{k+1})f(x_{k+1}^p) + \sum_{j=1}^{m-1}\beta_j^c(h_{k+1})f_{k-j+1}\}$
  Estimate Error Err
  if $Err > \epsilon$ **then**
    Recompute h^{err}, GOTO "Select Step Size"
  end if
  if $g(x_{k+1}, t_{k+1}) \geq -\epsilon_g$ **then**
    Event Reset $t_0 = t_{k+1}$, $x_0 = x_{k+1}$, $k = 0$, $i = i'$. GOTO "Mode Set".
  else
    Update $k++$, recompute h^{err}
  end if
until $t_k \geq t^{\text{final}}$

Fig. 5. Extrapolation-based event detection algorithm for unilateral event functions.

tical considerations and an example which constructs an extrapolation polynomial. We assume the user has subroutines capable of: estimating the truncation error; selecting a step size, h^{err}, based on this error; and solving for the roots of a polynomial. We also assume the user employs some standard bootstrapping procedure for estimating the previous m values of the state derivative during the initial m iterations.

Note that since it is impossible to guarantee $g(x,t) = 0$ exactly, $\epsilon_g \geq 0$ is a user defined termination tolerance. Also, note that the smallest positive real root of

the extrapolation polynomial, called h^* determines the most imminent event and should dictate the step size. If there are no positive real roots, $h^* = \infty$, meaning that *for the purposes of event detection* there is no upper limit on the step size. However, in practice, event considerations are not the only criteria which determine the appropriate step size to be used in simulation. Often the high-level simulation algorithm will specify some minimum step size, $h_{\min}$ below which roundoff errors interfere with the computation. In addition, most modern numerical integrators estimate an ideal step size based on truncation error considerations, h^{err}. The reconciliation of these various possible step sizes is accomplished in the line titled "Select step size". The following example illustrates the construction of $P_k^i(h_{k+1})$, by fully expanding $\beta(h_{k+1})$ for a two step Adams method.

Example 2.8. Let the integration method be a two step Adams method, $\beta_1 = (1 + h_{k+1})/(2h_k)$ and $\beta_2 = -h_{k+1}/(2h_k)$. Substituting the expressions for β into eq. (15) gives

$$P_k^i(h_{k+1}) = h_{k+1}\left(\left[f_k\frac{(1+h_{k+1})}{(2h_k)} - f_{k-1}\frac{h_{k+1}}{(2h_k)}\right]\cdot\frac{\partial g}{\partial x} + \frac{\partial g}{\partial t}\right) + (1-\gamma)g_k^p.$$

Rearranging gives

$$P_k^i(h_{k+1}) = a_2h_{k+1}^2 + a_1h_{k+1} + a_0$$

where

$$\begin{aligned} a_2 &= \frac{1}{2h_k}[\partial g/\partial x\cdot(f_k - f_{k-1})],\\ a_1 &= \frac{1}{2h_k}\frac{\partial g}{\partial x}\cdot f_k + \frac{\partial g}{\partial t},\\ a_0 &= (1-\gamma)g_k^p. \end{aligned}$$

Further, if $g(x,t) = c_0 + c_1\cdot x + c_2t$, where $c_0, c_2 \in \mathbb{R}$ and $c_1 \in \mathbb{R}^N$. Then $\partial g/\partial x = c_1$ and $\partial g/\partial t = c_2$.

3. EXAMPLES AND EXPERIMENTS

In this section we illustrate the operation of the algorithm and compare it to other popular algorithms via three examples – each meant to illustrate a different attribute of the method. All implementations described here were done in Matlab 6.1 on a 1GHz PC.

A common shortcoming of naive event detection approaches is their inability to reliably detect events when the event function has an even number of sign changes

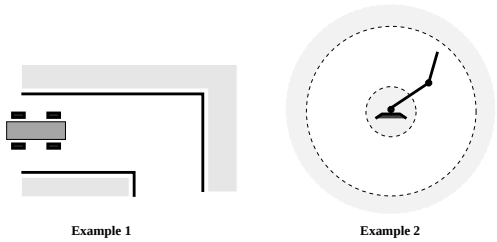

Fig. 6. Examples 1 and 2: (*left*) an autonomous vehicle navigating a corridor; (*right*) a planar two link robotic manipulator with workspace limitations. In the case of the vehcile, the state is (x, y), the location of its center of mass, and θ, its orientation measured counter-clockwise with repect to the positive x-axis. For the manipulator the state is θ_1, the orientation of the first link measured counter-clockwise with repect to the positive x-axis, and θ_2, the orientation of the second link measured counter-clockwise relative to the major axis of the first link.

on the integration interval. In Section 3.1 we compare the operation of the method introduced here to naive endpoint checking algorithms, establishing the basic competency of the algorithm. This places our algorithm among the ranks of other so-called "correct" algorithms, which by using interpolations are guaranteed to detect multiple events [Shampine et al. 1991; Park and Barton 1996; Bahl and Linninger 2001; Pritsker and O'Reilly 1999]. In Section 3.2 we look at an example with a model singularity. Indeed it is this type of situation which the method was designed to address. We compare the robustness of our extrapolation method to the interpolation-based methods. Our algorithm is clearly superior in this regard. Finally in Section 3.3 we compare the computational efficiency of our algorithm to that of the interpolation-based algorithms. Because interpolation based methods crash in situations with model singularities, which would prevent us from collecting meaningful computational statistics, this example is singularity free. Since *any* root finding scheme can be used with our method, we use implementations of both algorithms that are as similar as possible – highlighting their *intrinsic* differences.

Finally, note that regardless of the method used it does not make sense to locate the event with significantly greater accuracy than that which was used to generate the state estimates (see Shampine et al. [1991]). For this reason we do not compare the accuracy of any methods in pinpointing the location of events.

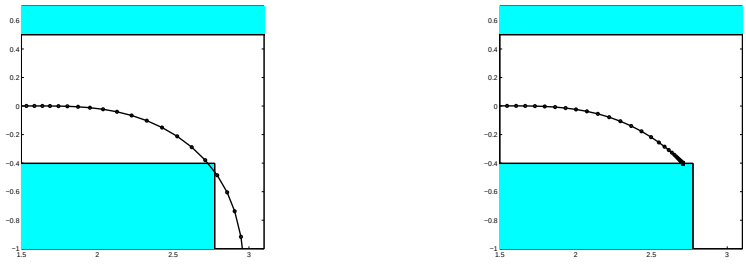

Fig. 7. Simulations of the mobile robot in Example 1. Dots indicate integrator output, while the solid lines connecting them are purely for visualization. (*left*) The endpoint checking technique fails to detect the collision; (*right*) the extrapolation based method reduces its step size as it approaches the event surface and properly detects the event.

3.1 Example 1: Reliability in Detecting Multiple Events

Consider the nonholonomic autonomous vehicle trying to navigate an indoor environment shown in Figure 6 (*left*). Assuming it is not in contact with a wall, the nominal kinematic equations are

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{22}$$

where the inputs are the forward velocity, $u_1 = 1$, and the turning rate, $u_2 = 0$ if $x < 1.75$ and $u_2 = 0.2$ if $x \geq 1.75$. Note that the inputs were selected so that the true solution could be easily calculated (a straight path followed by an arc). The goal here is to verify that the robot does not collide with the obstacles. For the sake of simplicity, we ignore the physical size of the robot and simply think of it as a point. Thus the collision event(s) for the simulation are given by the equations of the walls

$$((y - 0.5 \geq 0) \bigvee (x - 3.5 \geq 0)) \bigvee ((-y - 0.4 \geq 0) \bigwedge (2.8 - x \geq 0)). \tag{23}$$

For the sake of comparison, in addition to implementing our extrapolation based method, we implemented a Runge-Kutta variable step size integrator, which used simple endpoint checking to detect events. This method represents the entire class of naive, but commonly used, endpoint checking approaches. Both our extrapolation method and the Runge-Kutta were fourth order methods, using a variable step size and a desired relative truncation error tolerance of 10^{-4}. Figure 7 (*left*) displays

Table I. A Comparison of the Number of Events Properly Detected Using the Naive Endpoint Checking Algorithm vs. the Extrapolation Method.

Method used	End-point	Extrapolation
Events detected	32%	100%

a situation for which the endpoint checking algorithm fails. Integration points are computed which happen to land just outside the event surface. Thus the simulator detects no collision when in fact the robot has collided with the walls, near the corner ($x = 2.8$, $y = -0.4$). Figure 7 (*right*) illustrates the method presented in this paper. Observe how the integrator takes smaller step sizes as it approaches the event surface. Note that in this example the gain, $\gamma = 0.5$, was selected in such a way as to produce a very gradual slowdown, for the purposes of illustrating the technique. In practice, since the guards are linear, a gain of $\gamma = 0$ could have been used to force fast convergence.

This experiment was repeated 100 times, with small (radius of 0.05) uniform random perturbations of the initial conditions. The perturbations were such that all initial conditions should produce a collision (as determined by an analytical solution) by a small margin. Table I summarizes the outcome. The results support the earlier statement that the extrapolation method lies in the class of "correct" methods, such as Park and Barton [1996] or Shampine et al. [1991], or commercial packages such as *Arena* [2004] , which are guaranteed to detect an even number of sign changes in the event function.

3.2 Example 2: Robustness to Model Singularities

Consider the planar two link manipulator, as shown in Figure 6 (*right*), with the kinematic equations

$$\begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}. \tag{24}$$

Control of such systems is often done in a hierarchical manner where desired (x, y) positions for the end effector are fed to the motor controller from a high level planner and the model is required to calculate θ_1 and θ_2 to achieve these positions. If the length of the proximal link is l_1 and the distal link is l_2 where $l_1 > l_2$, the appropriate inverse kinematic relations to compute θ_1 and θ_2 as a function of (x, y)

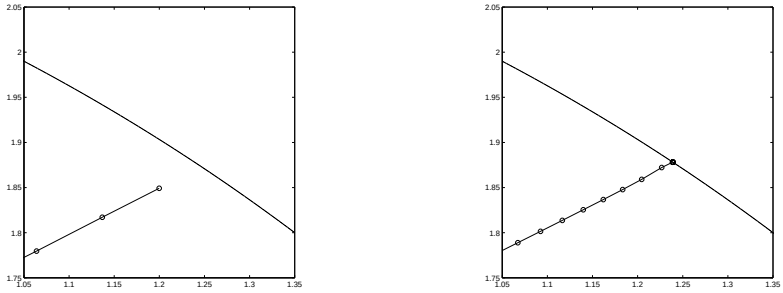

Fig. 8. Simulations of the two link robotic manipulator from Example 2: (*left*) interpolation-based methods cannot be used since the vector field is ill-defined out side the workspace; (*right*) the extrapolation based method approaches the surface asymptotically without ever requiring a function evaluation outside the workspace.

are

$$\theta_1 = \tan^{-1}(y/x) - \cos^{-1}\left(\frac{x^2+y^2+l_1^2-l_2^2}{2l_1\sqrt{x^2+y^2}}\right) \tag{25}$$

$$\theta_2 = \tan^{-1}\left[\frac{y-l_1\sin(\theta_1)}{x-l_1\cos(\theta_1)}\right] - \theta_1. \tag{26}$$

Typically the trajectories are time parameterized and $\omega_1(t)$ and $\omega_2(t)$ are calculated accordingly.

Note that it is possible for the high-level planner to be unaware of the specifics of the manipulator and specify (x, y) points which are outside the set of reachable positions of the manipulator, in such cases the arguments of the $\cos^{-1}$ function would fall outside of the range of $[-1, 1]$ and the right hand-side of the differential equation becomes ill-defined. In this case the event function would be

$$(\sqrt{(x^2+y^2)} \leq (l_1+l_2)) \bigwedge (\sqrt{(x^2+y^2)} \geq (l_1-l_2)) \tag{27}$$

with $x = l_1\cos\theta_1 + l_2\cos(\theta_1+\theta_2)$, $y = l_1\sin\theta_1 + l_2\sin(\theta_1+\theta_2)$. Figure 8 displays a simulation of the two-link manipulator attempting to track a reference trajectory, which is a straight line in Cartesian space. In this case the reference trajectory eventually falls outside the workspace of the manipulator, where the right-hand side of the differential equation becomes complex.

A fourth order predictor-correct integrator, with variable step size was used. The desired relative truncation error was 10^{-4}. For the sake of comparison we implemented our extrapolation method as well as an interpolation-based event detection method based on the natural corrector interpolant similar in spirit to that

of Shampine et al. [1991] or Park and Barton [1996]. As indicated in Figure 8 (*left*), the interpolation-based simulator generates a point near the edge of the workspace and its next point falls outside the workspace. Because the vector field is ill-defined there, it is unable to correctly compute this new point, nor is it able to activate its root finding algorithm since the construction of the interpolant requires an initial point on each side of the event surface. The output of our algorithm is shown in Figure 8 (*right*). Successively smaller steps are taken as the state approaches the boundary of the workspace. This experiment was repeated 100 times with random initial conditions and in each case the results were the same. It is important to note that the while the various interpolation-based algorithms differ from the one we implemented in how they compute the roots of the interpolation polynomial, they all share the same basic underlying algorithm and therefore would all fail in a similar manner in the presence of a model singularity.

3.3 Example 3: Computational Efficiency

In this section we run our extrapolation-based event detection method and the generic interpolation-based method, on a larger example problem in order to assess the computational efficiency of our algorithm. It is important to emphasize that, while both methods employ an integration scheme and a root-finding scheme, neither the interpolation-based nor our extrapolation-based method are tied to specific such algorithms. Indeed the extrapolation method can be used with any predictor-corrector method regardless of order, and it treats root-finding as a "black box" subroutine. Therefore, any method to determine the roots – such as the highly efficient "root exclusion test" used in Park and Barton [1996] – can be incorporated with either method. Our goal is to ascertain the quantity of additional computation time intrinsic to the extrapolation method. Again, both methods were implemented using fourth order predictor-corrector formula in Matlab with a relative truncation error tolerance of 10^{-4}, and each used Matlab's built-in root finding scheme `roots`.

We chose an event detection benchmark problem which possess no model singularities to prevent the interpolation-based methods from crashing during the experiment. The "bumper cars" problem consists of M vehicles with dynamic equations such as eq. (7) in a two-dimensional box (see Figure 9). We assume there are 20 cars, with radius $r = 0.1$ and the dimensions of the box is 10×10. A collision between two cars or a car and the wall constitutes an event. The post-collision

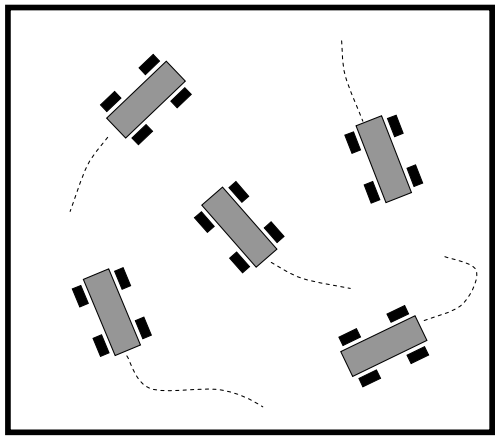

Fig. 9. Example 3: The bumper car problem.

Table II. A Comparison of the Average Computation Times of the Standard Interpolation Method and the Extrapolation Method Introduced Here. Both Methods Were Run on the "Bumper Car" Example Problem 100 Times with Random Initial Conditions on a 1GHz PC.

Method used	Interpolation	Extrapolation
Computation Time	0.3106 sec	0.3330 sec

velocities are computed assuming the collisions are inelastic and the coefficient of restitution is 0.9.

Again, 100 simulations were run, each with randomly generated initial conditions, using Matlab version 6.1 on a 1 GHz PC. The algorithms simulated 100 seconds of system dynamics. From Table II it appears that the price one pays for the added robustness of the extrapolation based method is that it requires, on average, 7.2% more computation time than the interpolation based method, for this example.

Recall that the major differences between the methods was:

1. reordering of the steps of the algorithms;
2. constructing extrapolation vs. interpolation polynomials; and
3. any additional integration steps required by the extrapolation due to the use of overly conservative small step sizes in the neighborhood of event surfaces.

Clearly item 1 has no effect on computation time. The difference due to item 2 is easily measured and is repeatable. By examining the cost of constructing each type of polynomial, clearly the major additional computation is in computing

the constant γ which ensures that the event surface is conservatively approached from one side. There is no equivalent quantity to be computed for the interpolation method. This accounts for approximately 6 floating point operations per integration step. The percentage of total computation time that accounts for per iteration depends on the cost of the function evaluations associated with the derivatives and event functions. In this example, the derivative functions and event functions were fairly simple, although there were many of them. By examining the average cost per iteration, we found that in this example the computation of γ accounted for 96% of the overall 7.2% discrepancy in computation time. The impact of item 3 is actually quite small, accounting for the remaining 4% of additional computation. This is explained by the fact that events are typically quite "rare," accounting for only 0.02% of all integration steps. We feel that this example is actually a conservative benchmark for assessing the efficiency of the method. Examples with more complex derivative functions or less frequent events would fare even better.

4. CONCLUSION

One of the most important facets of numerically simulating nonsmooth differential equations (a.k.a. hybrid or switched systems) is detecting, locating and properly handling discontinuities in the derivative of the state (a.k.a. state events). These events are signaled by the zeros of an event function. Since the underlying "solution" to the differential equations is not known analytically, but rather values of the state are generated at discrete points in time, it can be very difficult to properly detect the occurrence of such an event when there are an even number of zero crossings in a particular integration interval. Only a small subset of all event detection techniques guarantee that *all* events will be detected – these methods are referred to here as interpolation-based methods. However, even these otherwise successful methods can fail in the neighborhood of a model singularity – a region of the state space where the the derivative function is undefined. Such events have been referred to as *unilateral events* elsewhere in the literature. Indeed, often the reason non-smooth models are employed is that no single expression for the derivative is valid in all operating regimes. Systems with model singularities include logic-based feedback control systems, phase transition systems, and chemical kinetics, to name a few. Interpolation methods fail because they require a derivative evaluation at the endpoint – which potentially lies inside the singular region.

The event detection algorithm described here overcomes this limitation using an approach inspired by feedback control theory. A carefully constructed extrapolation polynomial is used to select the integration step size by checking for potential future events, avoiding the need to evaluate the differential equation in potentially singular regions. The types of event functions for which the step size can be computed exactly are functions with Taylor series expansions of finite length. This includes polynomial or linear time- and state-dependent event functions. In the case of arbitrary nonlinear functions there are two options. The nonlinear guards can be converted to linear form by appending an extra state variable (this requires symbolic manipulation capabilities); or an approximate linearization can be numerically computed using the first few terms of the inverse Taylor series expansion. In the case of guards which are exactly linearizable, it is possible to locate every event with accuracy on the same order as the integration accuracy. The user may specify on which side of the event surface the final value of the state should be computed.

Three examples are used to demonstrate the effectiveness and computational cost of the method. First we demonstrate that the extrapolation method is capable of detecting even numbers of event function zero crossings, a common shortcoming of some event detection methods. In this regard our extrapolation method is on equal footing with interpolation-based methods, lauded for their ability to properly handle this situation. Next we compare the extrapolation and interpolation methods in their ability to properly handle events near model singularities. The extrapolation method is clearly superior in this regard. It is this basic defect of interpolation methods that inspired the development of the new approach advocated here. Finally, we carefully benchmark the performance of our method comparing it to interpolation-based methods. In exchange for increased robustness, our method only requires a modest increase in computation time (approximately 7.2% more computation time for the example considered). Most of the discrepancy is accounted for by the added computation required to construct the extrapolation polynomial and the conservatism in step size selection required to avoid singularities. We feel that the chosen example actually understates the performance of the algorithm.

ACKNOWLEDGMENTS

The authors would like to thank George J. Pappas for helpful discussions in the early stages of this work, and the Editor-in-Chief for numerous suggestions that greatly improved the quality of this paper.

REFERENCES

Ascher, U. and Petzold, L. 1998. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Bahl, V. and Linninger, A. 2001. Modeling of continuous–discrete processes. In *Hybrid Systems: Computation and Control*, M. D. Benedetto and A. Sangiovanni-Vincentelli, Eds. Lecture Notes in Computer Science, vol. 2034. Springer Verlag, 387–402.

Benan, K., Campbell, S., and Petzold, L. 1989. *Numerical Solutions of Initial Value Problems*. North Holland, London.

Branicky, M. 1995. Studies in hybrid systems: Modeling, analysis and control. Ph.D. thesis, MIT, Cambridge, MA.

Carver, M. 1978. Efficient integration over discontinuities in ordinary differential equation simulations. *Mathematics and Computers in Simulation XX*, 190–196.

Cellier, F. 1979. Combined discrete/ continuous system simulation by use of digital computers: techniques and tools. Ph.D. thesis, ETH Zurich, Zurich, Switzerland.

Esposito, J. M. 2002. Simulation and control of hybrid systems with applications to mobile robotics. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Esposito, J. M. and Kumar, V. October 2004. An asynchronous integration and event detection algorithm for simulating multi-agent hybrid systems. *ACM Transactions on Computer Modeling and Simulation 14,* 4, 336–358.

Gustafsson, K. 1994. Control-theoretic techniques for stepsize selection in implicit runge-kutta methods. *ACM Transactions on Mathematical Software 20,* 4, 496–517.

Hay, J. and Griffin, A. 1979. Simulation of discontinuous dynamical systems. In *Proceedings of the 9th IMACS Conference on Simulation of Systems*. 79–87.

Isidori, A. 1995. *Nonlinear Control Systems*. Springer, London.

Joglekar, G. and Reklaitis, G. 1984. A simulator for batch and semi-continuous processes. *Computers in Chemical Engineering 8,* 6, 315–327.

Kelton, W., Sadowski, R., and D.T.Sturrock. 2004. *Simulation with Arena. 3rd ed.* Boston: McGraw Hill.

Mosterman, P. 1999. An overview of hybrid simulation phenomena and their support by simulation packages. In *Hybrid Systems : Computation and Control*, F. Vaandrager and J. H. van Schuppen, Eds. Lecture Notes in Computer Science, vol. 1569. Springer Verlag, 163–177.

Park, T. and Barton, P. 1996. State event location in differential-algebraic models. *ACM Transactions on Modeling and Computer Simulation 6,* 2, 137–165.

Prestin, A. and Berzine, M. 1991. Algorithms for the location of discontinuities in dynamic simulation problems. *Computers in Chemical Engineering 15,* 10, 701–713.

Pritsker, A. and O'Reilly, J. 1999. *Simulation with Visual SLAM and AweSim. 2nd ed.* New York: John Wiley and Sons.

Ruohonen, K. 1994. Event detection for ODEs and nonrecuirsive heirarchies. In *Results and trends in theoretical computer science*. Lecture Notes in Computer Science, vol. 812. Springer Verlag, 358–371.

Shampine, L., Gladwell, I., and Brankin, R. March 1991. Reliable solution of special event location problems for ODEs. *ACM Transactions on Mathematical Software 17,* 1, 11–25.

Tomlin, C. and Greenstreet, M., Eds. 2002. *5th International Workshop, Hybrid Systems: Computation and Control*. Springer LNCS 2289, Palo Alto, CA.